Algoritmi e strutture dati
Complementi di algoritmi

Roberto Cordone

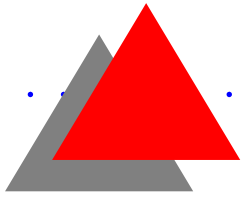
DTI - Università degli Studi di Milano

Polo Didattico e di Ricerca di Crema

Tel. (0373 / 898) **054**

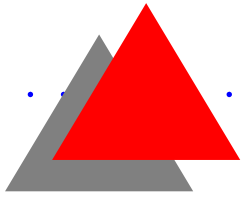
E-mail: **cordone@dti.unimi.it**

Web page: **http://www.dti.unimi.it/~cordone**



Scopo del laboratorio

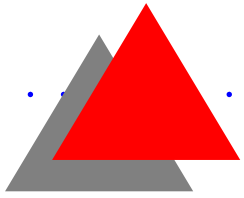
- Implementare in pratica strutture dati e algoritmi
- Ragionare sui fondamenti della programmazione
- Abituarsi al rigore, per poter affrontare programmi
  - di grandi dimensioni
  - scritti in collaborazione



Modalità del laboratorio

- Problema concreto da risolvere con un algoritmo
- Modellazione: dal problema alle strutture dati
- Progetto dell'algoritmo risolutivo
- Scomposizione in fasi; per ognuna
  - definizione di poche semplici operazioni
  - stesura del codice (lavoro libero)
  - lettura e commento della soluzione

Nelle prime lezioni, rapidi richiami di sintassi C



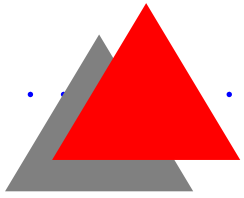
Risultati attesi

Acquisire uno stile nella scrittura del codice

- Non regole ferree, ma. . .
- principi generali basati su motivazioni ragionate
- convenzioni pratiche autoimposte, sempre migliorabili e aperte alla critica costruttiva

Siete invitati a

- propormi regole alternative
- (se non mi convincete) adeguarvi alle mie regole

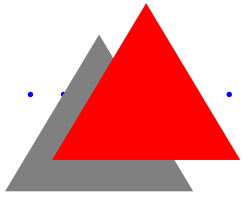


Codice ben scritto

Un codice ben scritto è modulare, cioè

esplicitamente diviso in procedure che si scambiano quantità limitate di informazioni esplicitamente elencate

- riutilizzare parti di codice la cui funzione è generale
- visualizzare a colpo d'occhio ogni singola procedura
  (non dovrebbe mai superare un paio di schermate)
- limitare ed esplicitare le informazioni scambiate fra i moduli
  (meglio evitare le variabili globali)
- facilitare l'individuazione dei moduli
  - scorretti (grazie al debugger e alla relazione ingresso-uscita)
  - inefficienti in tempo (grazie al profiler)
  - inefficienti in spazio (grazie al monitor di sistema)

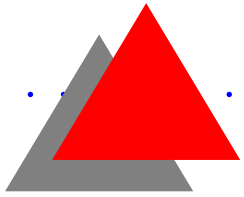


Dal semplice al complesso

- prima i sottoproblemi meno complessi e meno importanti
- prima le procedure più semplici ed esterne
  (lettura dati, scrittura risultati, allocazioni e deallocazioni. . .)

Perché?

- isolare il problema dall'interfaccia col mondo esterno
- isolare il nucleo specifico del problema da sottoproblemi meno specifici
- verificare continuamente la correttezza
  - nella sintassi (ricompilando)
  - nella semantica (verificando il legame ingresso-uscita su dati test)
- costringersi a ragionare in termini di specifiche
  (procedure = "scatole nere" con morsetti di ingresso e di uscita)



Verifiche

- Faremo subito tutti i controlli possibili (sensati)
  sulle fonti di errore (input/output, allocazioni, ecc. . .)
- Istituiremo messaggi di errori significativi e univoci, per capire dove e quando si verifica l'errore (e quindi forse perché)
- Procederemo dall'esterno verso l'interno, non dall'inizio alla fine
  - per ogni apertura di file, la corrispondente chiusura
  - per ogni inizio di ciclo o test, la corrispondente fine
  - per ogni allocazione, la corrispondente deallocazione
- Useremo il più possibile nomi simbolici
  (per non dover ricordare il significato di costanti numeriche)